# Capítulo 3: Funções

# Função

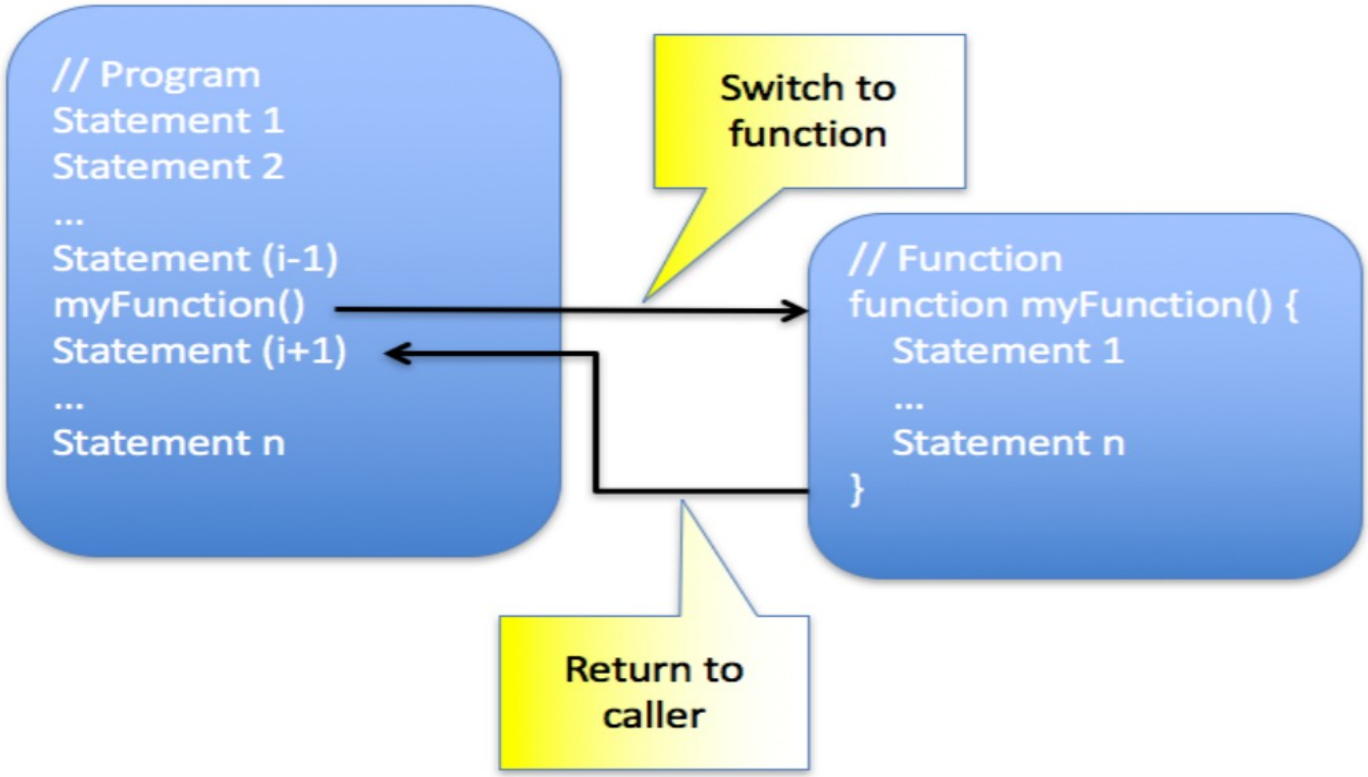
**Função é um "subprograma" que pode ser chamado por código externo (ou interno no caso de recursão) à função.**

**Assim como o programa em si, uma função é composta por uma sequência de instruções chamada *corpo da função*. Valores podem ser *passados* para uma função e ela vai *retornar um valor*.**

**Em JavaScript, funções são objetos de primeira classe, pois elas podem ter propriedades e métodos como qualquer outro objeto. O que as difere de outros objetos é que as funções podem ser chamadas.**

Em resumo, elas são objetos do tipo Function.

# Funções


// Program
Statement 1
Statement 2
...
Statement (i-1)
myFunction()
Statement (i+1)
...
Statement n
Switch to function
// Function
function myFunction() {
Statement 1
...
Statement n
}
Return to caller

# Declarando uma função

**Lembram-se dos passos para se trabalhar com variáveis (declarar, inicializar e usar)?**

**Pois as funções envolvem apenas dois momentos: declarar e usar (ou chamar)**

**A *declaração* (ou *definição)* consiste no uso da palavra chave *function*, seguida por:**

Nome da função, lista de parâmetros - entre parênteses, corpo da função - entre chaves { }.

Ex:

```
function explode(){
   return “buuuummmm!!!!”;
}
```

**As funções declaradas *não* são executadas imediatamente. Elas são "salvas para uso posterior" e serão executadas mais tarde, quando forem invocadas (chamadas).**

# Invocando (ou chamando) uma função

**O código dentro de uma função *não* é executado quando a função é *declarada*.**

**O código dentro de uma função somente é executado quando a função é *chamada***

**1) declara:**

```
function multiplica(a, b) {          //a e b são os parâmetros da função
  return a * b;
}
```

**2) chama:**

```
multiplica(10, 2);                   //10 e 2 são argumentos da função
```

# Função: cidadã de primeira classe

**1) Atribuir uma função a uma variável (expressão de função*):**

```
const foo = function() {
  console.log("foobar");
}
foo(); // Chamar a função usando a variável
```

***embora não obrigatório, essa função é normalmente anônima (usa-se o nome da variável para acessar-se a função)**

**2) Passar uma função como um argumento*:**

```
function oi() {
  return "Oi, ";
}
function cumprimentar(mensagem, nome) {
  console.log(mensagem() + nome);
}
// Passar  a função`oi` como um argumento pra função //
`cumprimentar`
cumprimentar(oi, "JavaScript!");
```

# Função: cidadã de primeira classe

**3)Uma função pode *conter* outra função (função aninhada):**

```
function somaQuadrados(a,b) {
  function quadrado(x) {
    return x * x;
  }
  return quadrado(a) + quadrado(b);
}
a = somaQuadrados(2,3); // retorna 13
```

**4)Uma função pode ser *retornada* por outra função*:**

```
function sayHello() {
  return function() {
    console.log("Hello!");
  }
}
sayHello()()
```

*** uma função que retorna outra é chamada de função de ordem superior (*higher order function*)**

# Expressão de função

**Funções também podem ser criadas por uma *expressão de função*. Tal função pode ser anônima:**

```
const quadrado = function(numero) {return numero * numero};   //Uma função armazenada em variável NÃO precisa de nome!
let x = quadrado(4) //x recebe o valor 16
```

**No entanto, um nome pode ser fornecido com uma expressão de função e pode ser utilizado no interior da função para se referir a si mesma (recursão):**

```
var fatorial = function fac(n) {return n<2 ? 1 : n*fac(n-1)};
console.log(fatorial(3));
```

# Passando função como argumento

**Além de dados, uma função Javascript pode também ter funções como parâmetro**

**Uma função passada como parâmetro é chamada *callback***

***O que se passa é a referência à função (ou seja, omitem-se os parênteses)***

*Ex:*

```
function alo(){
    console.log('alô, mundo');
}
setTimeout(alo, 3000);        //a função "alo" só será chamada passados 3 segundos
                               //setTimeout é uma função nativa do browser
```

# Funções aninhadas

**JavaScript permite uma função dentro de outra (aninhamento de funções)**

**A função interna tem acesso a todas as variáveis e funções definidas na função externa**

**O inverso não é verdade: a função externa não tem acesso às variáveis e funções definidas dentro da função interna**

```
function externa(){
    let x = 'alô, mundo';
    function interna(){          //função aninhada
        console.log(x);
    }
    Interna();
}
```

# Retornando uma função

**Uma função javascript também pode retornar uma função:**

```
function externa(){
    let x = 'alô, mundo';
    function interna(){
        console.log(x);
    }
    return interna;
}
```

# Recursão

**Recursão é a capacidade de uma função chamar a si própria**

**Para evitar um loop infinito, deve-se prever uma condição de saída**

```
function fatorial(n) {
    if(n<2) return 1;    //condição de retorno
    return n*fatorial(n-1);
}
console.log(fatorial(5)); // 1ª iteração retorna: 5*fatorial(4)
                          //2ª iteração retorna: 5*(4*fatorial(3))
                          //3ª iteração retorna: 5*(4*(3*fatorial(2)))
                          //4ª iteração retorna: 5*(4*(3*(2*fatorial(1))))
                          //5ª iteração retorna: 5*4*3*3*1)
```

# Convertendo um loop em Recursão

```
let x = 0;
while (x < 10) { // "x <10" é a condição do loop
  // faz algo...
  x++;
}
```

...pode ser convertido em uma declaração de função recursiva, seguida por uma chamada a essa função:

```
function loop(x) {
  if (x >= 10)  return;  //1.condição de saída
  // 2. faz algo...
  loop(x + 1); //3. chamada recursiva
}
 // "x> = 10" é equivalente a "! (x <10)"
```

# Arrow function

- **Introduzida no ES6.**
- **Permite escrever uma função de forma abreviada**
- **São funções anônimas**

**Ex:**

```
const oi = function(){
    console.log('oi');
}
```

**Vira:**

```
const oi = ()=>console.log("oi");
```

- **Como toda expressão de função, chama-se uma arrow function pela variável:**
  - oi()  //retorna "oi"
- **A função arrow pode ter parâmetros:**

**Ex:**

```
const soma(a,b) => a+b;
Soma(2,3) //5
```

- **Arrow functions não são içadas (hoisted)**

# Exercícios de funções

**1) Execute a função abaixo:**

```
function minhaFuncao() {
  alert("Alô, mundo!");
}
```

**2)**

**a) Crie uma função que retorne a string “alô, mundo”. Em seguida, chame-a armazenando seu retorno em uma variável. Por fim, imprima a variável no console.**

**b) Crie a função acima usando expressão de função. Em seguida, chame-a armazenando seu retorno em uma variável. Por fim, imprima a variável no console.**

# Exercícios de funções

3) Você sabe quantos anos seu cachorro tem em anos humanos, mas em anos de cachorro? Calcule!

Escreva uma função chamada calculaIdadeCanina que:

- aceite 1 argumento: a idade do seu cachorro.

- calcule a idade do seu cão com base na taxa de conversão de 1 ano humano igual a 7 anos para cães.

- exiba o resultado na tela assim: "Seu cachorrinho tem XX anos em anos de cachorro!"

Chame a função três vezes com diferentes valores.

4) Crie 2 funções que calculam propriedades de um círculo, usando as definições aqui:

a. Crie uma função chamada calcCircunferencia:

- Passe o raio para a função.

- Calcule a circunferência com base no raio e gere "A circunferência é XX".

b. Crie uma função chamada calcArea:

- Passe o raio para a função.

- Calcule a área com base no raio e gere "A área é XX".

# Exercícios de funções

**5) Crie uma função chamada celsiusFahrenheit:**

**- Armazene uma temperatura Celsius em uma variável.**

**- Converta-o para fahrenheit e gere "NN ° C é NN ° F".**

**b. Crie uma função chamada fahrenheitCelsius:**

**- Agora armazene uma temperatura fahrenheit em uma variável.**

**- Converta-o para celsius e exiba "NN ° F é NN ° C."**

**6) Fizz buzz é um jogo de grupo para as crianças aprenderem divisão. Os jogadores se revezam para contar de forma incremental, substituindo qualquer número divisível por três pela palavra "fizz", e números divisíveis por cinco pela palavra "buzz" e números divisíveis por 3 e 5 devem ser substituídos for "FizzBuzz". Qualquer outro número repete-se o próprio número.**

Ex: 1, 2, Fizz, 4, Buzz, Fizz, 7, 8, Fizz, Buzz, 11, Fizz, 13, 14, FizzBuzz, 16, 17

**Implemente esse jogo em Javascript.**